A Relation between a Group and a Ring (Computer Algebra : Algorithms, Implementations and Applications)

AUTHOR(S):

Kobayashi, Hidetsune; Suzuki, Hideo; Murao, Hirokazu

数理解析研究所講究録 1335 巻 2003 年 13-19

A Relation between a Group and a Ring

Hidetsune Kobayashi
Nihon University

Hideo Suzuki
Tokyo Institute, Polytechnic University

Hirokazu Murao
University of Electro-Communications

1 Introduction

We tried to prove abstract ring theory with Isabelle/HOL [1]. And we saw differences between expressions in a text book of abstract ring theory and Isabelle/HOL. In mathematics, an abelian group is defined as a group with a condition "commutativity of the binary operator", and the operator is traditionally expressed by + which may different from the originally adopted symbol expressing the binary operator. But in Isabelle in some situations, we cannot change the symbol. In Isabelle/HOL we have two ways to treat abelian groups. One is to use locale; we can define an abelian group locale with "commutativity" added to the assumption of a general group. But, in this case we have to use the symbol expressing the binary operator which is non-commutative. This gives an uncomfortable feelings when we treat a ring. Another way to treat an abelian group is to define abelian group independently to the general group, and make a bridge between a general group and an abelian group. The latter way is our main object to be discussed in this report.

2 A Definition of a Group

We first prepare a structure for groups consisting of a binary operation, an inverse operation and a unit for the operation:

```
record 'a grouptype =
  pcarrier ::  "'a set"
  bOp1     ::  "['a, 'a ] ⇒ 'a"
  iOp1     ::  "'a ⇒ 'a"
  unit1    ::  "'a" ("(υ_)" [81]80)
```

and then, we define a group by assigning required properties:

```
constdefs Group :: "('a, 'more) grouptype_scheme ⇒ bool"
  "Group G == (bOp1 G): pcarrier G → pcarrier G → pcarrier G ∧
    (iOp1 G) ∈ pcarrier G → pcarrier G ∧
    (unit1 G) ∈ pcarrier G ∧
    (∀x ∈ pcarrier G. ∀y ∈ pcarrier G. ∀z ∈ pcarrier G.
      (bOp1 G (unit1 G) x = x) ∧
      (bOp1 G (iOp1 G x) x = unit1 G) ∧
      (bOp1 G (bOp1 G x y) z = bOp1 G x (bOp1 G y z)))"
```

Using *locale* facility[2], we may define the same as follows:

```
record 'a grouptype =
  pcarrier :: "'a set"
  bop1     :: "['a, 'a ] ⇒ 'a" (infixl "⋆ı" 65)
  iop1     :: "'a ⇒ 'a" ("(_¯ı)" [81] 80)
  unit1    :: "'a" ("υı")
locale group = struct G +
assumes bop1_fun:  "bOp1 G ∈ pcarrier G → pcarrier G → pcarrier G"
    and iop1_fun:  "iop1 G ∈ pcarrier G → pcarrier G"
    and unit1_elm:  "υ ∈ pcarrier G"
    and bop1_assoc:"⟦ x ∈ pcarrier G; y ∈ pcarrier G; z ∈ pcarrier G ⟧ ⟹
                        (x ⋆ y) ⋆ z = x ⋆ (y ⋆ z)"
    and unit1_fx:"x ∈ pcarrier G ⟹ υ ⋆ x = x"
    and left_inv:"x ∈ pcarrier G ⟹ (x¯) ⋆ x = υ "
```

Locale facilitates defining local contexts, *i.e.*, locally fixed variables, local assumptions and local definitions. This is a convenient gear to treat an object extracted from the category of group. Since we have to treat the category of group with functor, we do not adopt locale. We use the former definition, and we introduce a symbol expressing the `bOp1` and the `iOp1` as

```
syntax
  "@BOP1" ::  "['a, ('a, 'more) grouptype_scheme, 'a] ⇒ 'a"
                                        ("(3 _/ ·-/ _)" [80,80,81]80)
  "@IOP1" ::  "['a, ('a, 'more) grouptype_scheme] ⇒ 'a"
                                        ("(_¯-)" [82,83]82)
translations
  "x ·G y" == "bOp1 G x y"
  "x^-G" == "iOp1 G x"
```

With these definitions and symbols above, we prove propositions on groups. For example, we give a simple proposition insisting that *the intersection of subgroups of a group is also a subgroup of the group*, as follows, where the notation $H \ll G$ is defined elsewhere and means H is a subgroup of G. The proof is almost automatic as shown below.

```
lemma inter_subgs:  "⟦ Group G; H ≪ G; K ≪ G ⟧ ⟹ (H ∩ K) ≪ G"
apply (simp add:Subgroup_def)
apply auto
done
```

3 A definition of an abelian group

In order for later expansion into rings, we want to use "+" as a symbol expressing a binary operator of abelian groups. For this purpose, we define an abelian group, independently from groups, as follows:

```
record 'a agrouptype =
  carrier ::  "'a set"
  abOp1   ::  "['a, 'a ] ⇒ 'a"
  aiOp1   ::  "'a ⇒ 'a"
  aunit1  ::  "'a"

constdefs AGroup ::  "('a, 'more) agrouptype_scheme ⇒ bool"
  "AGroup G == (abOp1 G): carrier G → carrier G → carrier G ∧
    (aiOp1 G) ∈ carrier G → carrier G ∧
    (aunit1 G) ∈ carrier G ∧
    (∀x ∈ carrier G. ∀y ∈ carrier G. ∀z ∈ carrier G.
        (abOp1 G (aunit1 G) x = x) ∧
        (abOp1 G (aiOp1 G x) x = aunit1 G) ∧
        (abOp1 G (abOp1 G x y) z = abOp1 G x (abOp1 G y z)) ∧
        (abOp1 G x y = abOp1 G y x))"
```

Types grouptype and agrouptype are similar but independent each other. Therefore, we cannot use lemmas obtained with a structure grouptype as those for abelian groups directly. We need a bridge between these two types. The numbers of lemmas and definitions for multiplicative groups are converted through the bridge lemmas and definitions for abelian groups. In some situations, we have to treat both multiplicative group and abelian group, and the bridge gives us a natural way to treat them together.

```
constdefs
  B_AG::"('a, 'more) agrouptype_scheme ⇒
         (| pcarrier::'a set, bOp1::['a, 'a] ⇒ 'a, iOp1::'a ⇒ 'a, unit1::'a |)"
  "B_AG G == (| pcarrier = carrier G, bOp1 = abOp1 G,
                iOp1 = aiOp1 G, unit1 = aunit1 G |)"
```

Using B_AG as a bridge between the grouptype and the abelian grouptype, we can induce lemmas for abelian groups from the lemmas obtained for the grouptype. And we can define mathematical concepts concerning abelian group by means of concepts already defined for general groups. For example, we define a subgroup of abelian group as:

```
constdefs
  ASubgroup ::  "[('a, 'more) agrouptype_scheme, 'a set] ⇒ bool"
  "ASubgroup G H == H ≪ (B_AG G)"   (* H≪G = H is a subgroup of G *)
```

and a definition of a normal subgroup is simply:

```
constdefs
  ANsubgroup ::  "[('a, 'more) agrouptype_scheme, 'a set] ⇒ bool"
  "ANsubgroup G N == N ◁ (B_AG G)"   (* H◁G = H is a normal subgroup of G *)
```

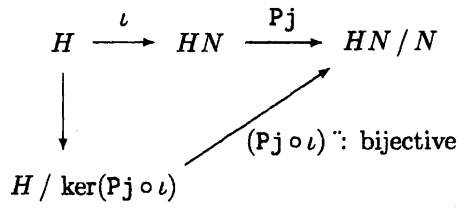
4 Groups and subgroups in Isabelle/HOL

In Isabelle/HOL a group is a structure having underlying set as the carrier, binary operator, inverse operator and unit as the components. Subgroup is a subset of the carrier, satisfying conditions. Subgroup and group has different structures. So, in Isabelle/HOL we cannot discuss whether two subgroups are isomorphic or not, provided the isomorphism means the group isomorphism as in mathematics. It seems to us a natural way to define a group structure induced to the subgroup from the group containing it:

```
constdefs Grp ::  "[('a, 'more) grouptype_scheme, 'a set ] ⇒ 'a grouptype"
  "Grp G H == (| pcarrier = H, bOp1 = λx∈H. λy∈H. (bOp1 G x y),
                 iOp1 = λx∈H. (iOp1 G x), unit1 = unit1 G |)"
```

The following from the homomorphism theory gives the second isomorphism theory, and demonstrates the Grp, where the group structure of HN denoted by $H \Diamond_G N$ is treated. We further use the following notations and definitions:

- for $f : F \to G$, $f^{\cdot\cdot}_{F,G} : F/\ker f \to G$,
- $f \circ_F g$ is a function composition of f and g where F is a range of f,
- Pj GH: $G \to G/H$;
- $\iota_F = \lambda x \in F.\ x$.




```
theorem homom4:
  "[| Group G; N ◁ G; H ≪ G |] ⟹
       bijec((Grp G H)/(H ∩ N)),((Grp G (H ◇G N))/N)
             (((Pj (Grp G (H ◇G N)) N) ∘(Grp G H) (ι(Grp G H)))
               ··(Grp G H), ((Grp G (H ◇G N))/N))"
apply (frule homom3Tr2 [of "G" "H" "N"], assumption+)
apply (frule subgns1, assumption+)
apply (frule homom4Tr1[of "G" "N" "H"], assumption+)
apply (frule subgGrp [of "G" "H"], assumption+)
apply (frule ind_hom_injec
             [of "Grp G H" "(Grp G (H ◇G N)/N)"
               "(Pj (Grp G (H ◇G N)) N) ∘(Grp G H) (ι(Grp G H))"],
       assumption+)
apply (simp add:surjec_def)
apply (frule inducedhomsurjec
             [of "Grp G H" "(Grp G (H ◇G N))/N"
               "(Pj (Grp G (H ◇G N)) N) ∘(Grp G H) (ι(Grp G H))"],
       assumption+)
apply (frule homom3Tr1[of "G" "H" "N"], assumption+)
apply simp
apply (simp add:bijec_def)
done
```

5 A definition of a ring

A definition of ring is as follows:

```
record 'a ringtype = "'a agrouptype" +
  bOp2  ::  "['a, 'a ] ⇒ 'a"
  unit2 ::  "'a"
```

```
constdefs
  Ring ::  "('a, 'more) ringtype_scheme ⇒ bool"
  "Ring R == AGroup R ∧(bOp2 R): carrier R → carrier R → carrier R ∧
    (unit2 R) ∈ carrier R ∧
    (∀x ∈ carrier R. ∀y ∈ carrier R. ∀z ∈ carrier R.
        (bOp2 R (unit2 R) x = x) ∧
        (bOp2 R x (abOp1 R y z) = abOp1 R (bOp2 R x y) (bOp2 R x z)) ∧
        (bOp2 R (bOp2 R x y) z = bOp2 R x (bOp2 R y z)) ∧
        (bOp2 R x y = bOp2 R y x))"
syntax
  "@BOP2" ::  "['a, ('a, 'more) ringtype_scheme, 'a] → 'a"
                        ("(3 _/ ·_/ _)" [85,85,86]85)
  "@AUNIT2" ::  "('a, 'more) ringtype_scheme → 'a"       ("'1_" [87]88)
translations
  "x ·R y" == "bOp2 R x y"
  "1R" == "unit2 R"
```

We define an ideal as:

```
constdefs
  ideal::"[('a, 'more) ringtype_scheme, 'a set] ⇒ bool"
    "ideal R I == I <+ R ∧ (∀r∈carrier R. ∀x∈I.(r ·R x ∈ I))"
```

6 What we have proved in Isabelle/HOL

The following lists what we have proved algready in Isabelle/HOL, giving sample descriptions of some typical theorems.

1. Jordan, Hoelder, Schreider theorem of a Group

```
lemma J_H_S: "[| Group G; Ugp E; compseries G r f; compseries G s g; 0 < r; 0 < s |]
                ⟹ r = s"
```

2. Proofs related to maximal ideals e.g.

   (a) existence of a maximal ideal (Zorn's lemma is written by Jacques D. Fleuriot)

```
(* existence of a maximal ideal *)
lemma id_maximal_Exist:  "[| Ring R; ¬(ZeroRing R) |] ⟹ ∃I. maximal_ideal R I"
```

```
(* existence of a maximal element disjoint from a multiplicatively closed set*)
lemma ex_mulDisj_maximal:
  "[| Ring R; mul_closed_set R S; 0_R ∉ S; 1_R ∈ S;
        T = { I. ideal R I ∧ S ∩ I = {} } |] ⟹ ∃mx. maximal_set T mx"
```

```
(* existence of a prime ideal *)
lemma ex_mulDisj_prime:"[| Ring R; mul_closed_set R S; 0_R ∉ S; 1_R ∈ S |]
        ⟹ ∃mx. prime_ideal R mx ∧ S ∩ mx = {}"
```

(b) a lemma on the set of nilpotent elements

```
(* a property of the nilradical *)
lemma nilradTr1:
  "[| Ring R; ¬ ZeroRing R |] ⟹ nilrad R = ⋂ { p. prime_ideal R p }"
```

(c) properties concerning Jacobson radicals

```
lemma J_rad_unit:  "[| Ring R; ¬ ZeroRing R; x ∈ J_rad R |]
        ⟹ ∀y. (y∈ carrier R ⟶ unit R (1_R +_R (-_R x) ·_R y))"
```

(d) elementary properties of local rings

```
lemma local_ring_diff:
  "[| Ring R; ¬ ZeroRing R; ideal R mx; mx ≠ carrier R;
      ∀a∈ (carrier R - mx). unit R a |] ⟹ local_ring R ∧ maximal_ideal R mx"
```

(e) properties related to residue class rings and a field

3. We can define a direct product of ideals parametrized by elements of a set A (not necessarily a finite set).

```
(* direct product of rings parametrized by a set A *)
constdefs
  Prod_ring::  "['a set, 'a ⇒ ('b, 'more) ringtype_scheme] ⇒
    (| carrier::('a ⇒ 'b) set, abOp1::['a ⇒ 'b, 'a ⇒ 'b] ⇒ ('a ⇒ 'b),
      aiOp1::('a ⇒ 'b) ⇒ ('a ⇒ 'b), aunit1::('a ⇒ 'b),
      bOp2 ::['a ⇒ 'b, 'a ⇒ 'b] ⇒ ('a ⇒ 'b), unit2::('a ⇒ 'b) |)"

  "Prod_ring A B == (| carrier = ac_Prod_Rg A B,
      abOp1 = λf. λg. prod_bOp1 A B f g, aiOp1 = λf. prod_iOp1 A B f,
      aunit1 = prod_unit1 A B, bOp2 = λf. λg. prod_bOp2 A B f g,
      unit2 = prod_unit2 A B |)"

translations "rΠ_A B" == "Prod_ring A B"
```

4. Chinese remainder theorem in general case:

$$R/\bigcap_{i=0}^{n+1} J_i \cong_r \prod_{i=0}^{n+1} R/J_i.$$

```
theorem Chinese_remThm:
  "[| Ring R; (∀k∈Nset (Suc n). ideal R (J k));
      ∀k∈Nset (Suc n). B k = QRing R (J k); ∀k∈Nset (Suc n). S k = pj R (J k);
      ∀i∈Nset (Suc n). ∀j∈Nset (Suc n). (i ≠ j ⟶ coprime_ideals R (J i) (J j)) |]
  ⟹ rbijec(QRing R (⋂{ I. ∃k∈Nset (Suc n). I = (J k)})),(rΠ(Nset (Suc n)) B)
       ((A_to_Prod (Nset (Suc n)) R S B)°R,(rΠ(Nset (Suc n)) B))"
```

References

[1] T. Nipkow, L. C. Paulson and M. Wenzel. *Isabelle's Logics:HOL.* `http://isabelle.in.tum.de/doc/logics-HOL.pdf`.

[2] F. Kammüller, M. Wenzel and L. C. Paulson. Locales: A Sectioning Concept for Isabelle. In *Theorem Proving in Higher Order Logics: TOHOLs '99*, LNCS 1690, Springer-Verlag, 1999.